```
 1
 2  struct Point
 3  {
 4      int x;
 5      int y;
 6
 7      Point()
 8          : x(0)
 9          , y(0)
10      {
11
12      }
13
14      Point(int x, int y)
15      {
16          this->x = x;
17          this->y = y;
18      }
19
20  };
21
22
23
24
```

```cpp
#include "PlaceableActor.h"

PlaceableActor::PlaceableActor(int x, int y, int colour)
    : m_pPosition(new Point(x, y))
    , m_IsActive(true),
    m_colour(colour)
{

}

PlaceableActor::~PlaceableActor()
{
    delete m_pPosition;
    m_pPosition = nullptr;
}

int PlaceableActor::GetXPosition()
{
    return m_pPosition->x;
}

int PlaceableActor::GetYPosition()
{
    return m_pPosition->y;
}

int* PlaceableActor::GetXPositionPointer()
{
    return &(m_pPosition->x);
}

int* PlaceableActor::GetYPositionPointer()
{
    return &(m_pPosition->y);
}

void PlaceableActor::SetXYPosition(int x, int y)
{
    m_pPosition->x = x;
    m_pPosition->y = y;
}

void PlaceableActor::Place(int x, int y)
{
    m_pPosition->x = x;
    m_pPosition->y = y;
    m_IsActive = true;
}
```

```h
#ifndef PLACEABLEACTOR_H
#define PLACEABLEACTOR_H

constexpr int kGreenColour = 10;
constexpr int kGreenColourSolid = 34;
constexpr int kRedColour = 12;
constexpr int kRedColourSolid = 14;
constexpr int kBlueColour = 9;
constexpr int kBlueColourSolid = 153;
constexpr int kRegularColour = 7;
#include "Point.h"

class PlaceableActor
{
public:
    PlaceableActor(int x, int y, int colour = kRegularColour);
    virtual ~PlaceableActor();

    int GetXPosition();
    int GetYPosition();
    int* GetXPositionPointer();
    int* GetYPositionPointer();
    void SetXYPosition(int x, int y);

    int GetColour() { return m_colour; }

    void Remove() { m_IsActive = false; }
    bool IsActive() { return m_IsActive; }
    void Place(int x, int y);

    virtual void Draw() = 0;
    virtual void Update() // some placeable actors will not need to update themselves
    {

    }

protected:
    Point* m_pPosition;
    bool m_IsActive;
    int m_colour;

};

#endif
```

```h
#include "PlaceableActor.h"

class Door : public PlaceableActor
{
public:
    Door(int x, int y, int colour, int closedColour);
    virtual void Draw() override;

    bool IsOpen() { return m_isOpen; }
    void Open() { m_isOpen = true; }

private:
    bool m_isOpen;
    int m_closedColour;

};
```

```cpp
#include <iostream>
#include <Windows.h>
#include "Door.h"

Door::Door(int x, int y, int colour, int closedColour)
    :PlaceableActor(x, y, colour)
    , m_isOpen(false)
    , m_closedColour(closedColour)
{};

void Door::Draw()
{
    HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
    if (m_isOpen)
    {
        SetConsoleTextAttribute(console, m_colour);
    }
    else
    {
        SetConsoleTextAttribute(console, m_closedColour);
    }
    std::cout << "|";
    SetConsoleTextAttribute(console, kRegularColour);
}
```

```h
#include "PlaceableActor.h"

class Enemy : public PlaceableActor
{
public:
    Enemy(int x, int y, int deltaX = 0, int deltaY = 0);
    virtual void Draw() override;
    virtual void Update() override;

private:

    int m_movementInX;
    int m_movementinY;

    int m_currentMovementX;
    int m_currentMovementY;

    int m_directionX;
    int m_directionY;

    void updateDirection(int& current, int& direction, int& movement);

};
```

```cpp
#include "Enemy.h"
#include <iostream>
#include <Windows.h>

Enemy::Enemy(int x, int y, int deltaX, int deltaY)
    : PlaceableActor(x, y, kGreenColour) // placing initial coordinates of enemy
    , m_currentMovementX(0)
    , m_currentMovementY(0)
    , m_directionX(0)
    , m_directionY(0)
    , m_movementInX(deltaX) //  The maximum distance the enemy can move in the x-direction
    , m_movementinY(deltaY) //  The maximum distance the enemy can move in the y-direction
{
    if (m_movementInX != 0)
    {
        m_directionX = 1;
    }
    if (m_movementinY != 0)
    {
        m_directionY = 1;
    }
}

void Enemy::Draw()
{
    HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(console, m_colour);
    std::cout << (char)153; // prints coloured enemy.
    SetConsoleTextAttribute(console, kRegularColour);
}

void Enemy::Update() // update the state of the enemy
{
    if (m_movementInX != 0)
    {
        updateDirection(m_currentMovementX, m_directionX, m_movementInX);
    }
    if (m_movementinY != 0)
    {
        updateDirection(m_currentMovementY, m_directionY, m_movementinY);
    }

    this->SetXYPosition(m_pPosition->x + m_directionX, m_pPosition->y + m_directionY);
}

void Enemy::updateDirection(int& current, int& direction, int& movement) // responsible for handling the movement of the enemy
{
    current += direction;
```

```
49      if (std::abs(current) > movement) // reverse movement. if we reach the
          end we want to loop back the other way.
50      {
51          current = movement * direction;
52          direction *= -1; // change direction
53      }
54  }
55
56  //  If the absolute value of the current movement becomes greater than the
      maximum allowed movement (movement), it means the enemy has reached the end
       of its allowed movement range
```

```cpp
#include "Money.h"
#include <iostream>

Money::Money(int x, int y, int worth)
    : PlaceableActor(x, y)
    , m_worth(worth)
{

}

void Money::Draw()
{
    std::cout << "$";
}
```

```h
#include "PlaceableActor.h"

class Money : public PlaceableActor
{
public:
    Money(int x, int y, int worth);

    int GetWorth() const { return m_worth; }

    virtual void Draw() override;

private:
    int m_worth;

};
```

```h
#include "PlaceableActor.h"

class Key : public PlaceableActor
{
public:
    Key(int x, int y, int colour)
        : PlaceableActor(x, y, colour)
    {
    }
    virtual void Draw() override;
};
```

```cpp
#include <iostream>
#include <Windows.h>
#include "Key.h"

void Key::Draw()
{
    HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(console, m_colour);
    std::cout << "+"; // prints coloured key.
    SetConsoleTextAttribute(console, kRegularColour);
}
```

```cpp
#include "Player.h"
#include "Key.h"
#include <iostream>

using namespace std;

constexpr int kStartNumberOfLives = 1;

Player::Player()
    : PlaceableActor(0,0)
    , m_pCurrentKey(nullptr)
    , m_money(0)
    , m_lives(kStartNumberOfLives)
{
};

bool Player::HasKey()
{
    return m_pCurrentKey != nullptr;
}

bool Player::HasKey(int colour)
{
    return HasKey() && m_pCurrentKey->GetColour() == colour;
}

void Player::PickUpKey(Key* key)
{
    m_pCurrentKey = key;
}

void Player::UseKey()
{
    m_pCurrentKey->Remove();
    m_pCurrentKey = nullptr;
}

void Player::DropKey()
{
    if (m_pCurrentKey)
    {
        m_pCurrentKey->Place(m_pPosition->x, m_pPosition->y);
        m_pCurrentKey = nullptr;
    }
}

void Player::Draw()
{
    cout << "@";
}
```

```
1  #ifndef _PLAYER_H_
2  #define _PLAYER_H_
3
4  #include "PlaceableActor.h"
5
6  class Key; // you can only forward declare pointer types
7
8  class Player : public PlaceableActor
9  {
10
11 public:
12     Player();
13
14     bool HasKey(); // Confusing
15     bool HasKey(int colour);
16     void PickUpKey(Key* key);
17     void UseKey();
18     void DropKey();
19
20     void AddMoney(int money) { m_money += money; }
21     int GetMoney() { return m_money; }
22
23     int GetLive() { return m_lives; }
24     void DecrementLives() { m_lives--; }
25
26     virtual void Draw() override;
27
28 private:
29     Key* m_pCurrentKey;
30     int m_money;
31     int m_lives;
32
33 };
34
35 #endif // !_PLAYER_H_
36
```

```
#include "Player.h"
#include <string>
#include <vector>
using namespace std;

class PlaceableActor;

class Level
{
    char* plevel;
    int height;
    int width;

    vector<PlaceableActor*> m_pActors;

public:
    Level();
    ~Level();

    bool LoadLevel(string levelName, int* playerX, int* playerY);
    void Draw();
    PlaceableActor* UpdateActors(int x, int y);

    bool IsSpace(int x, int y);
    bool IsWall(int x, int y);

    int GetHeight() { return height; }
    int GetWidth() { return width; }
    int GetIndex(int x, int y);

private:
    bool Convert(int* playerX, int* playerY);

};

```

```cpp
#include <Windows.h>
#include "Level.h"
#include <iostream>
#include <fstream>
#include "Player.h"
#include "Enemy.h" // derived Placeable Actor Classes
#include "Key.h"
#include "Door.h"
#include "Goal.h"
#include "Money.h"
#include <assert.h>

using namespace std;

constexpr char WAL = (char)219;

Level::Level()
    : plevel(nullptr)
    , height(0)
    , width(0)
{

};

Level::~Level()
{
    if (plevel != nullptr)
    {
        delete[] plevel;
        plevel = nullptr;
    }

    while (!m_pActors.empty())
    {
        delete m_pActors.back(); // return us the elements at end, then delete
        m_pActors.pop_back(); // continue to delete the remaining vector elements.
    }
};

bool Level::LoadLevel(string levelName, int* playerX, int* playerY)
{
    levelName.insert(0, "../");
    ifstream levelFile;
    levelFile.open(levelName);
    if (!levelFile)
    {
        cout << "An error has occured." << endl;
        return false;
    }
    else
    {
```

```
52          constexpr int tempSize = 25;
53          char temp[tempSize];
54
55          levelFile.getline(temp, tempSize, '\n');
56          width = atoi(temp); // converts integer into width.
57
58          levelFile.getline(temp, tempSize, '\n');
59          height = atoi(temp);
60
61          plevel = new char[width * height]; // array that we need to
              deallocate.
62          levelFile.read(plevel, width * height);
63
64          if (playerX != nullptr && playerY != nullptr)
65          {
66              bool anyWarnings = Convert(playerX, playerY);
67              if (anyWarnings)
68              {
69                  cout << "There are some warnings in the level data. see
                      above." << endl;
70                  system("pause");
71              }
72          }
73          return true;
74      }
75  }
76
77  void Level::Draw()
78  {
79      HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
80      SetConsoleTextAttribute(console, kRegularColour);
81
82      //Draw Level
83      for (int y = 0; y < GetHeight(); ++y)
84      {
85          for (int x = 0; x < GetWidth(); ++x)
86          {
87              int indexToPoint = GetIndex(x, y);
88              cout << plevel[indexToPoint];
89          }
90          cout << endl;
91      }
92
93      COORD actorCursorPosition; // position the cursor at correct location, x
           and y variables
94
95      // Draw actors
96
97      for (auto actor = m_pActors.begin(); actor != m_pActors.end(); +
          +actor) // going to the beginning and through the end.
98      {
99          if ((*actor)->IsActive()) // if active we want to draw.
100         {
```

```
101             actorCursorPosition.X = (*actor)->GetXPosition();
102             actorCursorPosition.Y = (*actor)->GetYPosition();
103             SetConsoleCursorPosition(console, actorCursorPosition); // set
                  position
104             (*actor)->Draw(); // draw the actors
105         }
106     }
107 }
108
109 bool Level::IsSpace(int x, int y)
110 {
111     return plevel[GetIndex(x, y)] == ' ';
112 }
113 bool Level::IsWall(int x, int y)
114 {
115     return plevel[GetIndex(x, y)] == WAL;
116 }
117
118 bool Level::Convert(int* playerX, int* playerY)
119 {
120     bool anyWarnings = false;
121
122     for (int y = 0; y < height; ++y)
123     {
124         for (int x = 0; x < width; ++x)
125         {
126             int intIndex = GetIndex(x, y);
127
128             switch (plevel[intIndex])
129             {
130             case '+':
131             case '-':
132             case '|':
133             {
134                 plevel[intIndex] = WAL;
135                 break;
136             }
137             case ' ':
138             {
139                 break;
140             };
141             case 'r':
142                 plevel[intIndex] = ' ';
143                 m_pActors.push_back(new Key(x, y, kRedColour));
144                 break;
145             case 'g':
146                 plevel[intIndex] = ' ';
147                 m_pActors.push_back(new Key(x, y, kGreenColour));
148                 break;
149             case 'b':
150                 plevel[intIndex] = ' ';
151                 m_pActors.push_back(new Key(x, y, kBlueColour));
152                 break;
```

```
153             case 'R':
154                 plevel[intIndex] = ' ';
155                 m_pActors.push_back(new Door(x, y, kRedColour,
                      kRedColourSolid));
156                 break;
157             case 'G':
158                 plevel[intIndex] = ' ';
159                 m_pActors.push_back(new Door(x, y, kGreenColour,
                      kGreenColourSolid));
160                 break;
161             case 'B':
162                 plevel[intIndex] = ' ';
163                 m_pActors.push_back(new Door(x, y, kBlueColour,
                      kBlueColourSolid));
164                 break;
165             case 'X':
166                 plevel[intIndex] = ' ';
167                 m_pActors.push_back(new Goal(x, y));
168                 break;
169             case '$':
170                 plevel[intIndex] = ' ';
171                 m_pActors.push_back(new Money(x, y, 1 + rand() % 5));
172                 break;
173             case '@':
174             {
175                 plevel[intIndex] = ' ';
176                 if (playerX != nullptr && playerY != nullptr)
177                 {
178                     *playerX = x;
179                     *playerY = y;
180                 }
181                 break;
182             }
183             case 'e':
184                 m_pActors.push_back(new Enemy(x, y));
185                 plevel[intIndex] = ' '; // clear level
186                 break;
187             case 'h': // horiztonal enemy
188                 m_pActors.push_back(new Enemy(x, y, 3, 0));
189                 plevel[intIndex] = ' ';
190                 break;
191             case 'v': // vertical enemy
192                 plevel[intIndex] = ' ';
193                 m_pActors.push_back(new Enemy(x, y, 0, 2));
194                 plevel[intIndex] = ' ';
195                 break;
196             default:
197             {
198                 cout << "Invalid character in file " << plevel[intIndex] <<
                      endl;
199                 anyWarnings = true;
200                 break;
201             }
```

```
202             }
203         }
204     }
205     return anyWarnings;
206 }
207
208 int Level::GetIndex(int x, int y)
209 {
210     return x + y * width;
211 }
212
213 // Updates all actors and returns a colliding actor is there is one.
214
215 PlaceableActor* Level::UpdateActors(int x, int y) // pass in x and y of
      player.
216 {
217
218     PlaceableActor* collidedActor = nullptr;
219
220     for (auto actor = m_pActors.begin(); actor != m_pActors.end(); ++actor)
221     {
222         (*actor)->Update(); //update all actors
223
224         if (x == (*actor)->GetXPosition() && y == (*actor)->GetYPosition
              ()) // collision occured
225         {
226             assert(collidedActor == nullptr); // cannot collide with
                  multiple actor
227             collidedActor = (*actor); // collided actor to the actors x and
                  y
228         }
229     }
230     return collidedActor;
231 }
```

```h
#include "Level.h"
#include "Player.h"


class Game
{
    Player player1;
    Level level1;
    bool gameOver;
    bool userQuit;

public:
    Game();
    ~Game();

    bool load();
    void Run();

    bool isGameOver();
    bool didUserQuit() { return userQuit; }
    int getPlayerLives() { return player1.GetLive(); }


private:
    bool Update();
    void Draw();

    bool HandleCollision(int newPlayerX, int newPlayerY);


};

```

```cpp
#include "Game.h"
#include <conio.h>
#include <Windows.h>
#include <iostream>

#include "Enemy.h"
#include "Key.h"
#include "Door.h"
#include "Money.h"
#include "Goal.h"

using namespace std;

constexpr int kArrowInput = 224;
constexpr int kLeftArrow = 75;
constexpr int kRightArrow = 77;
constexpr int kUpArrow = 72;
constexpr int kDownArrow = 80;
constexpr int kEscapeKey = 27;
constexpr int kBackspace = 8;

Game::Game()
    :gameOver{ false } {};

Game::~Game() {};

bool Game::load()
{
    return level1.LoadLevel("Level1.txt", player1.GetXPositionPointer(),
      player1.GetYPositionPointer());
}
void Game::Run()
{
    Draw();
    gameOver = Update();

    if (gameOver)
    {
        Draw();
    }
}

bool Game::isGameOver()
{
    return gameOver;
}

bool Game::Update()
{
    int input = _getch();
    int arrowInput = 0;
    int newPlayerX = player1.GetXPosition();
    int newPlayerY = player1.GetYPosition();
```

```
53
54      // One of the Arrow keys were pressed
55      if (input == kArrowInput)
56      {
57          arrowInput = _getch();
58      }
59
60      if ((input == kArrowInput && arrowInput == kRightArrow) ||
61          ((char)input == 'd' || (char)input == 'D'))
62      {
63          newPlayerX++;
64      }
65
66      if ((input == kArrowInput && arrowInput == kLeftArrow) ||
67          ((char)input == 'a' || (char)input == 'A'))
68      {
69          newPlayerX--;
70      }
71
72      if ((input == kArrowInput && arrowInput == kUpArrow) ||
73          ((char)input == 'w' || (char)input == 'W'))
74      {
75          newPlayerY--;
76      }
77
78      if ((input == kArrowInput && arrowInput == kDownArrow) ||
79          ((char)input == 's' || (char)input == 'S'))
80      {
81          newPlayerY++;
82      }
83
84      if (input == kEscapeKey)
85      {
86          userQuit = true;
87          return true;
88      }
89      if ((char)input == 'Z' || (char)input == 'z')
90      {
91          player1.DropKey();
92      }
93
94      //If position never changed
95
96      if (newPlayerX == player1.GetXPosition() && newPlayerY ==
          player1.GetYPosition())
97      {
98          return false;
99      }
100     else
101     {
102         return HandleCollision(newPlayerX, newPlayerY);
103     }
104 }
```

```
105
106 bool Game::HandleCollision(int newPlayerX, int newPlayerY)
107 {
108     PlaceableActor* collidedActor = level1.UpdateActors(newPlayerX,
        newPlayerY); // creates a placeable actor
109     if (collidedActor != nullptr && collidedActor->IsActive())
110     {
111         Enemy* collidedEnemy = dynamic_cast<Enemy*>(collidedActor); //
            specifies the type/ thing we are trying to cast, in this case an
            enermy
112         if (collidedEnemy)
113         { // if the pointer is valid, if statement works, if it is a key
            none of the code will work
114             collidedEnemy->Remove(); // if a collision with an enemy occurs,
                the enermy is removed.
115             player1.SetXYPosition(newPlayerX, newPlayerY); // players
                position is set to new position
116
117             player1.DecrementLives(); // decrmeent lives
118             if (player1.GetLive() < 0) // if less than zero game is over.
119             {
120                 return true; // game is over
121             }
122         }
123         Money* collidedMoney = dynamic_cast<Money*>(collidedActor); // if
            collided with money
124         if (collidedMoney)
125         {
126             collidedMoney->Remove(); // remove the money
127             player1.AddMoney(collidedMoney->GetWorth()); // add the money
                and show the worth.
128             player1.SetXYPosition(newPlayerX, newPlayerY);
129         }
130         Key* collidedKey = dynamic_cast<Key*>(collidedActor);
131         if (collidedKey)
132         {
133             if (!player1.HasKey())
134             {
135                 player1.PickUpKey(collidedKey);
136                 collidedKey->Remove();
137                 player1.SetXYPosition(newPlayerX, newPlayerY);
138             }
139         }
140         Door* collidedDoor = dynamic_cast<Door*>(collidedActor);
141         if (collidedDoor)
142         {
143             if (!collidedDoor->IsOpen())
144             {
145                 if (player1.HasKey(collidedDoor->GetColour()))
146                 {
147                     collidedDoor->Open();
148                     collidedDoor->Remove();
149                     player1.UseKey();
```

```
150                         player1.SetXYPosition(newPlayerX, newPlayerY);
151                 }
152                 else
153                 {
154 
155                 }
156             }
157             else
158             {
159                 player1.SetXYPosition(newPlayerX, newPlayerY); // player
                      goes through the door
160             }
161         }
162         Goal* collidedGoal = dynamic_cast<Goal*>(collidedActor);
163         if (collidedGoal)
164         {
165             collidedGoal->Remove(); // removes actors
166             player1.SetXYPosition(newPlayerX, newPlayerY);
167             return true;
168         }
169     }
170     else if (level1.IsSpace(newPlayerX, newPlayerY))
171     {
172         player1.SetXYPosition(newPlayerX, newPlayerY);
173     }
174     else if (level1.IsWall(newPlayerX, newPlayerY))
175     {
176         // wall collision
177     }
178     return false;
179 }
180 
181 void Game::Draw()
182 {
183     HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
184     system("cls");
185 
186     level1.Draw();
187 
188     //Set cursor position for player
189     COORD actorCursorPosition;
190     actorCursorPosition.X = player1.GetXPosition();
191     actorCursorPosition.Y = player1.GetYPosition();
192     SetConsoleCursorPosition(console, actorCursorPosition);
193     player1.Draw();
194 
195 
196     //Set cursor to end of level.
197     COORD currentCursorPosition;
198     actorCursorPosition.X = 0;
199     actorCursorPosition.Y = level1.GetHeight();
200     SetConsoleCursorPosition(console, actorCursorPosition);
201 }
```

```cpp
#include <iostream>
#include <conio.h>
#include <Windows.h>
#include <fStream>
#include "Game.h"
using namespace std;

int main() {

    Game myGame;

    if (myGame.load())
    {
        while (!myGame.isGameOver())
        {
            myGame.Run();
        }
        if (myGame.didUserQuit())
        {
            cout << "Thanks for playing!" << endl;
        }
        if (myGame.getPlayerLives() < 0)
        {
            cout << "You lose!" << endl;
        }
        else
        {
            cout << "You win!" << endl;
        }
    }
    else
    {
        cout << "Game did not load" << endl;
    }

}

//void PlayDoorClose()
//{
//  Beep(500, 75); // frequency and duration
//  Beep(500, 75);
//}
//void PlayerDoorOpen()
//{
//  Beep(1397, 97);
//}
//void PickUpKey()
//{
//  Beep(1568, 100);
//}
//void Win()
//{
//  Beep(1568, 200);
```

```
54  //  Beep(1568, 200);
55  //  Beep(1568, 200);
56  //  Beep(1245, 1000);
57  //  Beep(1397, 200);
58  //  Beep(1397, 200);
59  //  Beep(1397, 200);
60  //  Beep(1175, 1000);
61  //}
```